

Packet Switching System

This invention relates to packet switching systems (also known as cell switching systems) for communications networks, in particular methods for allocating
5 output switching requests for traffic from the inputs of a packet switch to its outputs. Fixed data units (slots) for switching are created by processing input packets as necessary or by any other means such as forecasting.

Input-buffered packet switches and routers offer potentially the highest available bandwidth for any given fabric and memory technology, but such systems
10 require accurate scheduling to make the best use of this bandwidth. In such a scheduling process the header of each incoming packet is processed to identify its destination and the individual packets are then buffered in corresponding input queues, one for each possible pairing of input port with output port (port pair). The scheduling process itself then determines a permutation (a switch configuration or an
15 input/output switch port assignment) in which packets from the input queues should be transmitted such that conflicts do not occur, such as two packets from different inputs competing for the same slot in the output ports.

On each cycle, many scheduling algorithms only check the occupancy of the first queue position (head of line) of each queue, but some check the occupancy of a
20 group of queue positions (known as "frames") as this is more efficient. A frame-based scheduler determines, in one process, a set of switch permutations (one permutation per slot duration) in the next frame period (the processed/scheduled frame).

Scheduling is one of the most serious limiting factors in Input-Buffered
25 packet switches. Scheduling generally consists of two sub-processes, namely matching (or arbitration), and time slot assignment (or switch fabric path-search). Matching is essentially the selection of packets from the input queues to maximise throughput within the constraints of frame lengths within both input and output ports (the "no-overbooking" constraint). Time slot assignment is the generation of a set of
30 permutations (switching matrix configurations) for routing the packets (slots) through the switch for each slot duration.

A suitable scheduling process must satisfy two conditions; firstly the matching process must ensure that a "no overbooking" criterion is met for each input

port and each output port (the "matching" problem). In other words it must arrange that the number of packets to be handled by each port (input and output), will not exceed the frame length (in number of time-slots) during the duration of the frame: ideally it should equal the frame length for each port, but this is not always possible as will be explained. Secondly, the time-slot assignment process must allocate all the matched requests for data units (time slots) for switching (in each permutation) during the frame time period. The present invention relates to the matching step of the scheduling process, which will be described in more detail later, after this overview of the whole packet switching system.

10 The packets are transmitted along a circuit established through the switch fabric according to a switching matrix (set of permutations) generated in the scheduling process just described. An output buffering stage may be provided in the output line cards before the packets are launched into the output links.

Maximum Size and Maximum Weight bipartite graph matching algorithms exist, which can theoretically achieve 100% throughput, but the complexity of their implementation makes them slow and their application unfeasible. However, although the Maximum Weight algorithm, and sometimes the Maximum Size algorithm, may solve the scheduling problem they are too complex to use on a per packet basis. There also exist more recent approaches due to Birkhoff and Von Neumann based on request matrix decomposition, but these generate long delays and thus are inappropriate for real-time scheduling. Hence, iterative, heuristic, parallel algorithms such as the i-SLIP process [N.McKeown, "The i-SLIP scheduling algorithm for input-queued switches", *IEEE Transactions on Networking*, vol. 7, no. 2, April 1999] and the Frame-based process have been developed. These heuristic algorithms are faster but, in the nature of heuristic algorithms, they do not provide a rigorous solution.

25 The "i-SLIP" scheduling algorithm, is an example of one that may operate slot-by-slot, (i.e. with a frame length of a single data unit or time-slot) but alternatively it may use a frame-based approach where input queues' occupancies are each checked once every F time-slots, where the value of F is greater than one: this interval of F timeslots is known as a "frame". The result of the scheduling process is a $N \times F$ Switch Matrix C , where N is the number of the switch input ports, from which switching configurations (set of permutations) are decided for the next frame-switching time period. The content of each element $c(i,s)$ of the matrix C is the

Switch Fabric Output Port number to which the "s"th slot of the frame coming from the "i"th input port is to be routed. Note that some elements in matrix C may be empty. Typically there are the same number of output ports as there are input ports (N). This is always likely to be the case unless there is a preponderance of one-way communications connections served by the switch.

The scheduling problem will now be described in more detail.

After the required output port of each incoming packet is identified, by processing the header or otherwise, the individual packets are buffered in corresponding input queues depending on the particular requested input/output port pair (VOQ). In order to establish the number of packets (time-slots), a counter is established for each queue (VOQ). Referring to Figure 1, which shows a typical switch system, the Switch Fabric 20 has N input ports 31...3N (labelled input I_1 to input I_N) and N output ports (labelled output O_1 to output O_N). The switching is under the control of a scheduler 10. In respect of each input port "i" the scheduler 10 maintains N queues (one per Output port "j"), labelled $VOQ_{i,j}$ in Figure 1, in which data units (slots) destined for the respective output port are buffered. Therefore in total there are N^2 Virtual Output queues, and N^2 counters.

The number of switching Requests, for each Input port/ Output port pair are stored in an $N \times N$ Request Matrix R. Each element $r(i,j)$ of this matrix shows the total number of packets pending in the VOQ between input port 'i' and output port 'j'.

In the example to be described below with reference to Figure 1, a switch fabric with $N=4$ is used for simplicity, but in a typical switch the value of N is a much larger number. A switching-time period (period for which permutations are decided) is for the duration of one frame (F slots), which can be one or more slots. This means that the matrix R is updated once per frame time-period (with the intention that as many as possible of the packets represented therein, according to the maximum switch capacity, will be switched during the following time period).

Outlets 'j' → Inlets 'i' ↓	O_1	O_2	O_3	O_4
I_1	3	4	2	0
I_2	5	0	1	0
I_3	8	5	1	3
I_4	2	0	2	6

$$R = \begin{pmatrix} 3 & 4 & 2 & 0 \\ 5 & 0 & 1 & 0 \\ 8 & 5 & 1 & 3 \\ 2 & 0 & 2 & 6 \end{pmatrix}$$

(1)

Matrix (1) represents a Request Matrix that will be used in this example, which has no further purpose other than to illustrate the scheduling process. Note that the total number of buffered packets for each port varies, in this example, between six (input I_2 and also output O_3) and eighteen (output O_1), and cannot
 5 therefore match the frame size for all ports. Therefore, either some packets will not be switched, (the data either being discarded or held over to the next frame), or some slots will be unused as there are not enough packets to use them all. In general the frame size is predetermined or could vary for each frame-period. Nonetheless it will be fixed for the duration of a frame scheduling.

10 In the matching process, a number of packets "F" corresponding to the frame length, selected from packets buffered at each input port queue are checked for acceptance, to make sure that there is no overbooking of the input and output ports within the frame. An NxN "Accepted-Requests Matrix" A is defined, whose elements $a_{i,j}$ represent the number of packet switching requests that are accepted
 15 from input port 'i' destined for output port 'j' in the next time period. The two conditions that ensure no overbooking are simply:

$$\sum_{j=1}^N a_{i,j} \leq F \quad \text{for all } i, \text{ and } \sum_{i=1}^N a_{i,j} \leq F \quad \text{for all } j.$$

Packets destined for overbooked ports may be discarded, or they may continue to be queued for transmission in later frames, if accuracy is more important
 20 than latency (delay time).

From the matrix R discussed above, the Matching process populates an NxN Accepted-Requests Matrix A. The values of the elements in this matrix are such that the switch input and output ports capacity is not exceeded, i.e. none of the row and column summations in this matrix exceeds F, which is the number of time slots (data
 25 units) that will be switched during the following time period. Note that we can generalise this definition to all Matching Algorithms, whether frame-based or not, saying for example that in a slot by slot process, such as the i-SLIP algorithm, the value of F is unity.

Matrix A below is an illustrative solution to the problem for the requests
 30 matrix R from (1). With a switching fabric with N=4 and a frame length F=8, the total switch capacity is NxF time-slots per time-period, which in this example is 32.

$\begin{matrix} \text{Outlets } j' \rightarrow \\ \text{Inlets } i' \downarrow \end{matrix}$	O_1	O_2	O_3	O_4	$\text{Row-Sums} \downarrow$
I_1	2	4	2	0	8
I_2	3	0	1	0	4
I_3	2	3	1	2	8
I_4	1	0	2	5	8
$\text{Column-Sums} \rightarrow$	8	7	6	7	$\text{Total Sum} = 28$ $N \times F = 32$

$$A = \begin{pmatrix} 2 & 4 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 2 & 3 & 1 & 2 \\ 1 & 0 & 2 & 5 \end{pmatrix} \quad (2)$$

A Matching algorithm attempts to completely fill up the matrix A taking the requests from matrix R, in such a way that the total switching requests for every input and output port does not exceed the value F (No-overbooking), and the total capacity of the Switch ($N \times F$) is achieved. We see that the example shown here has not achieved filling the matrix A to the maximum switch capacity (32 requests), but only 28. In fact, because the input queues are not balanced, it is not possible to fill the remaining four slots in this example.

The solution represented by illustrative matrix A could be achieved reasonably quickly by trial and error from the matrix R. However, in practice switch fabrics have values for N (the number of ports) much greater than the illustrative value of 4 and a systematic approach is required. The present invention presents such an approach. Before the invention is discussed, there follows a description of the subsequent stages in the process.

Usually, the Matching Algorithms only check the occupancy (inspecting the counters) of the first locations in each virtual output queue, or input-output pair queue (heads of queues) to a maximum of F locations per input port (i.e. all virtual queues corresponding to the same input port), without taking into account all switching requests. There exist some Matching Algorithms that check the occupancy of higher number of queue locations, for instance a multiple of the value of F. For example, if $F=8$, we could check occupancy in the first 16 locations ($2F$) in each backlogged input queue to try to completely populate the Accepted-Requests Matrix A.

On the other hand, some Matching Algorithms use a number of iterations. This means that part of the algorithm is run more than once, always using the same queue locations as in the first time. This usually improves the filling ratio of the Accepted-Requests Matrix and therefore the switching throughput. Examples include i-SLIP ($i > 1$), or Frame-based algorithms using some variants of the port pointer

update rule [A. Bianco et al., "Frame-based scheduling algorithms for best-effort cell or packet switching", Workshop on High Performance Switching and Routing, HPSR 2002, May 2002, Japan]. Different versions of the frame-based algorithm have different rules for updating the port pointers and some variants of the frame-based algorithm look twice at the buffer occupancies on the same locations. For example, the versions known as NOB-27 and NOB-25 both use the same update rule but NOB-27 runs part of the process twice on the same buffer locations.

Once the accepted requests matrix A has been generated, the second sub-process in the scheduling algorithm computes the set of switch permutations and assigns the time-slots within a frame to the accepted requests for each one of the permutations. In this way the switch fabric can be configured for each time slot avoiding conflicts at the output ports, i.e. there is at most one packet from any input queue to one particular output port. This process can be referred to as Time Slot Assignment (TSA). From the matrix of accepted requests A, we build the NxF Switch Matrix C. The elements $c(i,s)$ of C show the output port number to which a switching request in input port 'i' will be switched in the slot 's' of the frame of length F that is being scheduled. There are a number of algorithms to achieve this, among them the one described in the applicant's existing patent application WO01/67802. Any particular packet should be capable of transmission across the switch fabric during any one of the time slots in the frame, although normally packets from the same queue (that is to say, between the same pair of ports) would be transmitted in the same order that they had originally arrived at the input port.

For example, from the illustrative "accepted requests" matrix A found in (2) the Switch Matrix C shown in (3) might be generated. The columns of matrix C represent the time-slots. At each time-slot the switch fabric has to be configured such that the packets present at the Input Ports are connected to the Output Port shown in each element $c(i,s)$ of matrix C.

Time-slot 's' → Inlet 'i' ↓	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
I_1	1	1	2	2	2	2	3	3
I_2	3		1		1			1
I_3	2	2	4	4	3	1	1	2
I_4	4	3	3	1	4	4	4	4

$$C = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 \\ 3 & & 1 & & 1 & & & 1 \\ 2 & 2 & 4 & 4 & 3 & 1 & 1 & 2 \\ 4 & 3 & 3 & 1 & 4 & 4 & 4 & 4 \end{pmatrix} \quad (3)$$

Therefore matrix C shows a set of possible switch fabric configurations for an entire frame period. Each column of matrix C shows a switch permutation with no

output port conflicts, i.e., no column of matrix C contains more than one occurrence of any output port number. The matrix C is the final result of the entire scheduling problem. Note that where the frame size $F = 1$, as for the "i-SLIP" algorithm, matrix C is a column vector ($N \times 1$ matrix), and therefore the time-slot scheduling algorithm is
5 straightforward.

An output memory stage may be provided where the slots could be re-sequenced (re-ordering and/or closing gaps between slots belonging to the same original packet).

Scheduling is therefore made up of the matching problem, and the time slot
10 assignment problem. In the matching problem switching requests are accepted in such a way that the switching capacity is not exceeded while achieving maximum throughput. The assignment problem selects a set of switch fabric configurations (permutations) within the frame-length. This has known exact solutions at acceptable complexities. However, some issues might arise due to slot sequencing that could
15 lead to the necessity for an output memory stage where slots could be re-sequenced.

Although some hitherto known heuristic matching algorithms generally perform well, there are some situations in which their performance deteriorates, in particular with unbalanced traffic patterns and bursty traffic sources. The consequences are that these matching algorithms are unable to ensure the ideal
20 100% throughput, and therefore some packets will be dropped (lost). This type of situation is becoming increasingly significant, as it is foreseen that the network churn (traffic patterns) will become highly dynamic, and not predictable.

The performance degradation occurs because these algorithms check the queues' occupancy only within a limited number of locations in the input queues.
25 Therefore when a particular input queue backlog keeps growing these algorithms do not necessarily serve it because they are unaware of such condition. This leads to inefficiency becoming apparent at high traffic loads for highly unbalanced traffic patterns, that is, when the number of packets requesting to be switched for one particular or more input-output port pairs is much higher than for the others. This is
30 common to many prior art heuristic matching algorithms as this process only scans the occupancy of a specified limited number of locations in the backlogged input queues, eventually leading to some packets being processed too late in certain traffic conditions.

One aspect of the present invention seeks to overcome this weakness by splitting the problem into a number of stages. Another aspect applies a transformation process to the switching request matrix, factorising it with respect to the switch port capacities. Although this transformation process could be used on its own, it is preferably used as the initial stage of two or more stages according to the first aspect.

According to the present invention there is provided a method of allocating switch requests within a packet switch, the method comprising the steps of

- (a) generating switch request data for each input port indicative of the output ports to which data packets are to be transmitted ;
- (b) processing the switch request data for each input port to generate request data for each input port-output port pairing; and
- (c) generating an allocation plan by reducing the number of queue requests relating to each of one or both sets of ports by a value such that the number of requests relating to each member of the set or sets of ports is no greater than a predetermined frame value.

This process may be used as the initial stage in the invention disclosed in the applicant's co-pending International application, filed on the same date as the present application with Applicant's reference A30156 WO, and claiming priority from United Kingdom Applications 0218565.0 and 0228904.9. The claims of that application provide a method of allocating switch requests within a packet switch, the method comprising the steps of

- (a) generating switch request data for each input port indicative of the output ports to which data packets are to be transmitted ;
- (b) processing the switch request data for each input port to generate request data for each input port-output port pairing; and
- (c) generating an allocation plan for the switch for a frame of a defined number of packets, by a first stage in which allocation rules are applied such that the number of requests from each input port and to each output port is no greater than the defined frame length, and one or more further stages in which allocation rules are applied to allocate requests remaining unallocated by the previous stage.

Each stage attempts a complete solution to maximising allocations, using the unallocated requests remaining from the previous stage. The stages may use the

same or different allocation rules. Some of stages may arrive at their complete solutions by an iterative process, such as the NOB-27 process already referred to.

The transformation of the request data may be done by summing up the switching requests from each input port, or the switching requests to each output port, or both, and reducing the number of requests from each input port, and to each output port, in such cases where the number of requests is greater than the maximum capacity of the relevant port, by a factor selected such that the total number of requests from the corresponding input port or to the corresponding output port is no greater than the maximum capacity of the corresponding input port and the corresponding output port. Thus the queue with the greatest number of switching requests is identified and served so as to keep the packet switch in a stable state for any possible traffic pattern, provided the traffic is admissible, i.e., the average switch request rate to any output port does not exceed the line rate of that output port (this condition applies to any scheduling algorithm). This invention allows the matching process to achieve maximum possible throughput for any input traffic statistics and with any traffic pattern, at a low complexity.

The reduction of the request data may comprise reducing the number of requests in the input ports; and then reducing the number of requests in the resulting transformed request data where it still exceeds the capacity of the output ports. Alternatively the output ports may be considered before the input ports.

Alternatively, the reduction of the request data from each input port and to each output port may be done using a common factor selected such that the number of requests from each input port and to each output port is no greater than the maximum capacity of either port. This process is quicker, but may lose some possible allocations.

This process ensures that all queued requests are considered, and is computationally simple, and therefore relatively fast. However, it may leave some capacity unfilled, or cause unnecessary delays. It is preferably followed by one or more other allocation processes to fill any remaining capacity.

Unallocated switch requests may be reserved for use in the next stage of switch request allocation, or abandoned if they have an expiry time.

The invention extends to a method of packet switching wherein the packets are switched on the basis of the allocated routing, and to a packet switch in which

the input port-output port routing is allocated in accordance with the method of the invention, and packets are switched from an input port to a specified output port in accordance with the allocated routing.

An embodiment of the invention will now be described, by way of example,
5 with reference to the drawings, in which

Figure 1, which has already been discussed, illustrates a simplified packet switching system;

Figure 2 is a graph comparing the performance of the i-slip algorithm, a frame-based algorithm using the NOB25 pointer update rule as described in an earlier
10 patent application of the applicant (WO01/67803), and a two-stage process in which the first stage comprises a request data reduction process using a common factor. The second stage is the frame-based algorithm (NOB25) also used in the earlier patent application.

The matching process according to the present invention applies multiple
15 stages. Traditional heuristic matching processes (e.g. i-SLIP and Frame-based) find matrix A directly from R ,

$$R \Rightarrow A$$

The reduction of request data of the second aspect of the invention can be used on its own, but preferably precedes a heuristic matching algorithm, in general
20 partially populating matrix A . In such a two-stage process, the original request matrix $R_0 = R$ is transformed to find a "normalised matrix" R_{norm} , in which the capacities of the input and output ports are not exceeded, and a "remaining request" matrix R_1 , where $R_1 = R_0 - R_{norm}$. The matrix R_{norm} is used to start to populate the Accepted-Requests Matrix A . The partially populated matrix will be referred to as
25 A^- .

$$A^- \equiv R_{norm}$$

Now, to fill up the remaining capacity in the matrix A , it is necessary to use another matching algorithm. We could call this A^+ matrix,

$$R_1 \Rightarrow A^+$$

30 Now,

$$A = A^- + A^+$$

The matching matrix A is the sum of the two matrices found during a two-stage example of the present invention. A request matrix transformation may be applied to either stage, (preferably the first) applying to the second stage any other known matching algorithm, or it may be applied to both the first and second stages.

- 5 In general, the transformation presented here will precede the application of other matching algorithms.

Note that this splitting process can be reiterative in more than two stages, in each stage applying any transformation of this invention or any other known matching algorithm.

- 10 An example of the process of generating the transformed matrix, referred to herein as "Normalisation1", and the subsequent stages, will now be discussed, using an exemplary Request matrix:

$\begin{array}{c} \text{Outlets 'j' } \rightarrow \\ \text{Inlets 'i' } \downarrow \end{array}$	O_1	O_2	O_3	O_4	Row - Sums \downarrow
I_1	3	4	2	0	9
I_2	5	0	1	0	6
I_3	8	5	1	3	17
I_4	2	0	2	6	10
Column - Sums \rightarrow	18	9	6	9	Highest sum $mval = 18$

- In this example there are, as before, four input ports and four output ports,
15 and the frame length F is again 8. A value $mval$ is derived from this matrix, which is the largest sum of any column or any row in the matrix, which in this case is the total number of requests for output port 1.

The process requires the transformation of the request matrix R_0 to find a matrix R_{norm} , in which the capacities of the input and output ports are not exceeded.

- 20 The result is used to generate the partially filled matrix A^- . The transformation performed in this example uses a common factor $d = F / \max(F, mval)$, which in the case of this example is $d = 8/18$ or 0.44. In other words, if the number of requests in respect of any port -input or output- ($mval$) is greater than the frame length, the value of every term in the matrix is reduced by a factor such that the total number of
25 requests in respect of that port is equal to, or less than, the frame length.

$$R_0 = \begin{pmatrix} 3 & 4 & 2 & 0 \\ 5 & 0 & 1 & 0 \\ 8 & 5 & 1 & 3 \\ 2 & 0 & 2 & 6 \end{pmatrix} \Rightarrow R_{norm} = \left\lfloor \frac{8}{18} R_0 \right\rfloor = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \\ 3 & 2 & 0 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix} = A^-$$

and the remaining Request Matrix is

$$R_1 = R_0 - R_{norm} = \begin{pmatrix} 2 & 3 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 5 & 3 & 1 & 2 \\ 2 & 0 & 2 & 4 \end{pmatrix}$$

5 It will be seen that matrix A is not full yet. The remaining capacity in matrix A can then be filled using the updated matrix R_1 and for example the known Frame based algorithm of the applicant's existing International Patent Application WO01/67803 using the pointer update rule NOB25, or the process described in the applicant's International Patent application filing on the same date as the present
10 case and having agents' reference A30137WO and claiming priority from United Kingdom applications 0218565.0 and 0228903.1.

Further refinements of the invention will now be described.

The process "Normalisation 1" described above does not completely populate the matrix A , because of the generalisation of ' $mval$ ' to all elements in the matrix R_0 .
15 As a result, some Inlet-Outlet pairs in the R_0 matrix are not heavily loaded, and therefore could be allowed a smaller ' $mval$ ', i.e., a bigger transforming factor.

Limiting each column total to the frame length ensures that the "no overbooking" condition is met for each Output Port. Similarly, limiting each row total to the frame length analogously ensures "no overbooking" of all input ports. The
20 following variant embodiment generates a separate value ' $mval$ ' for each term in the matrix, being the highest of three values: the respective totals for the row and column of which that term is a member, and the frame length. We therefore have a set of normalising factors that may be different for each element of matrix R_0 . WE arrange this set in a matrix form. This transformation is referred to herein as
25 "Normalisation2".

$\begin{array}{c} \text{Outlets 'j' } \rightarrow \\ \text{Inlets 'i' } \downarrow \end{array}$	O_1	O_2	O_3	O_4	Row-Sums \downarrow	
I_1	3	4	2	0	9	$\Rightarrow mval = \begin{pmatrix} 18 & 9 & 9 & 9 \\ 18 & 9 & 6 & 9 \\ 18 & 17 & 17 & 17 \\ 18 & 10 & 10 & 10 \end{pmatrix}$
I_2	5	0	1	0	6	
I_3	8	5	1	3	17	
I_4	2	0	2	6	10	
Column-Sums \rightarrow	18	9	6	9		

In order to find each element $r_{norm\ ij}$ of the normalised matrix R_{norm} we have

$$r_{norm\ ij} = \left[r_{0\ ij} \cdot \frac{F}{\max(F, mval_{ij})} \right]$$

- 5 where $r_{0\ ij}$ represent each element of the request matrix R_0 , and $mval_{ij}$ represent each element of the matrix $mval$. This operation results in the following normalised matrix

$$\Rightarrow R_{norm} = \begin{pmatrix} 1 & 3 & 1 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 2 & 0 & 1 \\ 0 & 0 & 1 & 4 \end{pmatrix} = A^-$$

- 10 It will be seen that in this particular example the highest summation (' $mval$ ') is that for column O_1 , namely 18, and so all terms in the first column of the matrix are multiplied by $8/18 = 0.44$ (all decimal figures approximated to two places). The next highest summation is in row I_3 ($mval=17$), so all terms in row I_3 , except the first, are multiplied by $8/17 = 0.47$. Similarly all terms in row I_4 except the first are
- 15 multiplied by $8/10 = 0.8$, the remaining terms in row I_1 and columns O_2 and O_4 are all $8/9 = 0.89$. Finally, Row I_2 and column O_3 both have an ' $mval$ ' of 6, which is less than the frame length, so the term at the intersection of that row and column takes the value of unity (not $8/6$).

The Remaining Switch Request Matrix R_1 is then determined as:

$$R_1 = R_0 - R_{norm} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 3 & 0 & 0 & 0 \\ 5 & 3 & 1 & 2 \\ 2 & 0 & 1 & 2 \end{pmatrix}$$

In this case, we have been able to populate matrix A more densely than
5 using the "Normalisation 1" process described above, leaving less work-load for the
following stage. However, there is added complexity as now we need N^2 registers to
store the 'mval' matrix, instead of a single value.

A further development, referred to herein as "Normalisation3" consists in
including a further phase (or step) and simplifying the first one within the
10 transformation process. In this embodiment, the matrix R_0 is transformed using a
vector in a first step and in a second step transforming only one of the ports. The
vector can be derived from the 'mval' of each individual row or, as shown below,
each individual column, but otherwise follows the same procedure as previously
described.

$\begin{array}{c} \text{Outlets 'j' } \rightarrow \\ \text{Inlets 'i' } \downarrow \end{array}$		O_1	O_2	O_3	O_4	$\Rightarrow mval^1 = (18 \ 9 \ 6 \ 9)$
I_1		3	4	2	0	
I_2		5	0	1	0	
I_3		8	5	1	3	
I_4		2	0	2	6	
Column-Sums \rightarrow		18	9	6	9	

$$r_{norm\ ij}^1 = \left[r_{0\ ij} \cdot \frac{F}{\max(F, mval_j^1)} \right]$$

where $r_{0\ ij}$ represent the element in row i and column j of the request matrix R_0 , and
 $mval_j^1$ represent each element of the vector $mval^1$, j being the same value taken for
the element $r_{0\ ij}$. Therefore each matrix element $r_{0\ ij}$ belonging to the same column j
20 will be normalised using the same factor resulting from $F / \max(F, mval_j^1)$. This
operation results in the first normalised matrix of the process that is

15

$$\Rightarrow R_{norm}^1 = \begin{pmatrix} 1 & 3 & 2 & 0 \\ 2 & 0 & 1 & 0 \\ 3 & 4 & 1 & 2 \\ 0 & 0 & 2 & 5 \end{pmatrix}$$

After this first step is finished, the second step considers each individual column or row, whichever was not done in the first step. Any of these which exceed the maximum capacity are transformed again, but they are otherwise left as they are.

- 5 In this example, it can be seen from inspection of the third row of R_{norm}^1 that the request sum for input port 3 is still higher than the port capacity F :

		<i>Outlets 'j' →</i>				
		<i>Inlets 'i' ↓</i>				
		O_1	O_2	O_3	O_4	<i>Row – Sums ↓</i>
$R_{norm}^1 =$	I_1	1	3	2	0	6
	I_2	2	0	1	0	3
	I_3	3	4	1	2	10 > F
	I_4	0	0	2	5	7
<i>Column – Sums →</i>		6	7	6	7	

- From the matrix R_{norm}^1 we can find again a vector with the summing of each
10 matrix row. Therefore

$$R_{norm}^1 \Rightarrow mval^2 = (6 \ 3 \ 10 \ 7)$$

Now we can apply again the normalisation, that is

$$r_{norm\ ij}^2 = \left[r_{norm\ ij}^1 \cdot \frac{F}{\max(F, mval_i^2)} \right]$$

- 15 where $r_{norm\ ij}^1$ represent the element in row i and column j of the request matrix R_{norm}^1 , and $mval_i^2$ represent each element of the vector $mval^2$, i being the same value taken for the element $r_{norm\ ij}^1$. Hence each matrix element $r_{norm\ ij}^1$ belonging to the same row i will be normalised using the same factor resulting from $F / \max(F, mval_i^2)$. Therefore we perform a further step in this stage, in which only the requests in the port where
20 they exceed F (port capacity) are normalised by a factor $F / mval_i^2 = 8/10$ (note that in

this particular case $i=3$), remaining the rest of the elements unchanged. This operation results in the second and final normalised matrix of the process that is

$$R_{norm}^2 = \begin{pmatrix} 1 & 3 & 2 & 0 \\ 2 & 0 & 1 & 0 \\ 2 & 3 & 0 & 1 \\ 0 & 0 & 2 & 5 \end{pmatrix} = A^-$$

$$5 \quad R_1 = R_0 - R_{norm}^2 = \begin{pmatrix} 2 & 1 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 5 & 2 & 0 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix} \text{ Remaining Switch Request Matrix}$$

This algorithm can also be started using the Input requests summations, and reducing the output requests in the second stage, instead of the other way round as described above.

10 Instead of reducing each value in a row or column by a common factor, a common value could instead be subtracted.

Figure 2 shows a comparison of the mean packet delay for three processes:

1. a prior art system known as i-SLIP with 3 iterations, labelled 3-SLIP in the figure,
2. the prior art Frame-based matching algorithm (WO01/67803) previously
- 15 discussed, using the pointer update rule NOB25, labelled NOB25 in the Figure.
3. the present invention, labelled NOBITO in the figure, using a Normalisation process similar to Normalisation 2, except that a single mval, equal to the largest sum = of any row or column, was used for all terms in the matrix in the first stage and the same prior art Frame-based matching algorithm (with pointer update rule NOB25)
- 20 as a second stage.

The Frame-based matching algorithm was run using one iteration and a 32 time-slot duration frame in all cases. The scenario is a 8x8 switch, using bursty packet arrivals with a mean burst duration of 256 packets, and with a traffic matrix P, in which each element $P(i,j)$ indicates the probable level of traffic between input

25 port "i" and output port "j":

$$P = \frac{1}{255} \begin{pmatrix} 1 & 2 & 4 & 8 & 16 & 32 & 64 & 128 \\ 2 & 4 & 8 & 16 & 32 & 64 & 128 & 1 \\ 4 & 8 & 16 & 32 & 64 & 128 & 1 & 2 \\ 8 & 16 & 32 & 64 & 128 & 1 & 2 & 4 \\ 16 & 32 & 64 & 128 & 1 & 2 & 4 & 8 \\ 32 & 64 & 128 & 1 & 2 & 4 & 8 & 16 \\ 64 & 128 & 1 & 2 & 4 & 8 & 16 & 32 \\ 128 & 1 & 2 & 4 & 8 & 16 & 32 & 64 \end{pmatrix}$$

The results of Figure 2 are for the links represented by the antidiagonal terms of this matrix.

Figure 2 shows that the prior art systems are only capable of achieving a 90% throughput, while using the present embodiment it is able to achieve 100% throughput. Because the buffer lengths have to be finite, packets are dropped (lost) from the queues when they reach a maximum delay. This is shown in the graph, where the curves become horizontal.

Therefore, the invention shows all advantages of the i-SLIP and frame-based algorithms and dramatically improves the performance at high traffic loads for any type of traffic sources and traffic patterns. Following the first stage of the process, the second stage of the matching problem deals with the remaining request matrix filling in the rest of the slot switch capacity, using for example a single iteration of a frame-based algorithm.

Table 1 below presents a number of examples of the use of the present invention. Two different normalisation methods according to the present invention are compared. Normalisation method 3 assigns a separate '*mval*' in the row and column for each $r_{i,j}$ matrix entry. This means that some matrix entries could be rounded down twice. Normaliation 2 assigns the larger of the row and column '*mval*' for each matrix entry. By assigning only one '*mval*' to each matrix entry, each one is only rounded down once.

These normalisation processes are each shown in combination with three different second stages: namely those disclosed in the applicant's co-pending applications referred to above, A30137 and WO01/67803, (the latter using the NOB-25 rule), and another algorithm known as "Ring" which was proposed by Politecnico di Torino within the European Union's collaborative project DAVID ["Description of

Network Concepts and Frame of the Work", DAVID (IST-1999-11742) project Deliverable D111, March 2002]. This 'Ring' algorithm is a greedy maximal approximation of a maximum weight matching [R.E. Tarjan, "Data Structures and Network Algorithms", Society for Industrial and Applied Mathematics, November 5 1983], a well known problem in graph theory.

Comparative data is also shown for a single stage process, and for processes having two similar stages.

The resulting Accepted-Requests Matrices A for each combination shown in Table 1 are shown in Table 2. In this example it is seen that the examples using a preliminary stage of the normalisation process 2 (examples d,e, and f) and Normalisation Process 3 (examples g, h, and i) of the present invention provide a higher filling cardinality than those which do not. Of those, the Frame-based algorithm (using NOB25 rule) process (examples f and i) generate a larger number of filled requests, (up to 28) but the filled matrices of the "Ring" process, (examples d and g) and of the applicant's co-pending application A30137 referred to above (examples e and h) provide a better match to the proportions of the original Request matrix R_0 , i.e. closer to a maximum weight matching.

	Stage 1	Stage 2	Cardinality (No of Accepted Requests)
a)	(No first stage)	A30137	23
b)	Ring	Ring	25
c)	A30137	A30137	23
d)	Normalisation2	Ring	26
e)	Normalisation2	A30137	26
f)	Normalisation2	WO01/67803	28
g)	Normalisation3	Ring	27
h)	Normalisation3	A30137	27
i)	Normalisation3	WO01/67803	28

20 Table 1. Comparison of different combinations of algorithms in a two-stage implementation of the present invention.

Table 2. Accepted-Requests Matrices, using the different combinations of Table 1.

$A =$		
a)	b)	c)
$\begin{bmatrix} 0 & 4 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 0 & 4 & 2 & 0 \\ 2 & 0 & 1 & 0 \\ 6 & 2 & 0 & 0 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 0 & 4 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 8 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 \end{bmatrix}$
d)	e)	f)
$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 2 & 0 & 1 & 0 \\ 5 & 2 & 0 & 1 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 2 & 0 & 1 & 0 \\ 5 & 2 & 0 & 1 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 3 & 2 & 1 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix}$
g)	h)	i)
$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 3 & 0 & 1 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 4 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 4 & 3 & 0 & 1 \\ 0 & 0 & 2 & 6 \end{bmatrix}$	$\begin{bmatrix} 2 & 4 & 2 & 0 \\ 3 & 0 & 1 & 0 \\ 2 & 3 & 1 & 2 \\ 0 & 0 & 2 & 6 \end{bmatrix}$